

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平7-271760

(43) 公開日 平成7年(1995)10月20日

(51) Int.Cl.⁶

G 0 6 F 17/12
15/16
17/16

識別記号

3 9 0 Z

片内整理番号

F I

技術表示箇所

G 0 6 F 15/ 324

15/ 347

K

審査請求 未請求 請求項の数 4 O L (全 18 頁)

(21) 出願番号

特願平6-62241

(22) 出願日

平成6年(1994)3月31日

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中1015番地

(72) 発明者 中西 誠

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(74) 代理人 弁理士 小笠原 吉義 (外2名)

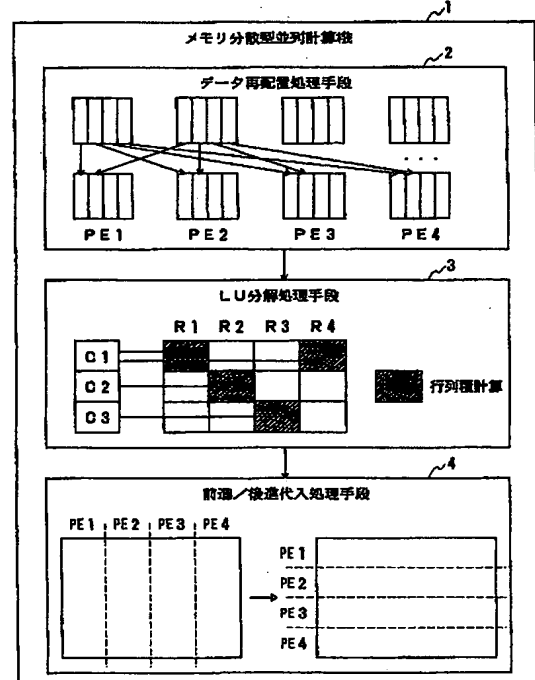
(54) 【発明の名称】 メモリ分散型並列計算機による連立1次方程式計算処理方法および計算機

(57) 【要約】

【目的】ブロック化した外積型のLU分解法により連立1次方程式を解くメモリ分散型並列計算機による連立1次方程式計算処理方法および計算機に関し、各プロセッサのLU分解の負荷を均等にするとともに、実質的なデータ転送コストを削減し、並列性を高めて高速化を図ることを目的とする。

【構成】データを列ベクトルを束ねたブロックレベルでサイクリックな配置に並列転送で並べ換え、LU分解時に行列積の計算対象となるデータを分割し、分割したデータに対する行列積の計算と、データ転送とを並列に同時に実行し、LU分解した結果について、データを元の配置に戻し、さらに行列を行ベクトル方向に分割した配置になるように並べ換え、並列に前進/後進代入の処理を行う。

本発明の原理説明図



【特許請求の範囲】

【請求項1】 複数のプロセッサを備え、各プロセッサ間でデータ転送を行うことのできるメモリ分散型並列計算機(1)を用い、係数行列を各プロセッサに分配して、ブロック化した外積型のLU分解法により連立1次方程式を解くメモリ分散型並列計算機による連立1次方程式計算処理方法において、前記メモリ分散型並列計算機の各プロセッサに分散されて配置されているLU分解の対象となる行列を、列ベクトルを束ねたブロックレベルでサイクリックな配置に並列転送でもって動的に配置しなおす処理過程と、各プロセッサに配置されたブロックについてLU分解を行う処理過程と、LU分解した結果について前進／後進代入処理を実行する処理過程とを有することを特徴とするメモリ分散型並列計算機による連立1次方程式計算処理方法。

【請求項2】 複数のプロセッサを備え、各プロセッサ間でデータ転送を行うことのできるメモリ分散型並列計算機(1)を用い、係数行列を各プロセッサに分配して、ブロック化した外積型のLU分解法により連立1次方程式を解くメモリ分散型並列計算機による連立1次方程式計算処理方法において、各プロセッサに配置されたブロックについてLU分解を行う途中で行列積の計算をするとき、行列積の計算対象となるデータを分割して各プロセッサに転送し、分割したデータに対する各プロセッサにおける行列積の計算と、他の分割したデータについての次の行列積の計算に用いる部分の並列転送とを同時に実行する処理を繰り返すことにより全体の計算を行う処理過程と、LU分解した結果について前進／後進代入処理を実行する処理過程とを有することを特徴とするメモリ分散型並列計算機による連立1次方程式計算処理方法。

【請求項3】 複数のプロセッサを備え、各プロセッサ間でデータ転送を行うことのできるメモリ分散型並列計算機(1)を用い、係数行列を各プロセッサに分配して、ブロック化した外積型のLU分解法により連立1次方程式を解くメモリ分散型並列計算機による連立1次方程式計算処理方法において、前記メモリ分散型並列計算機の各プロセッサに分散されて配置されているLU分解の対象となる行列を、列ベクトルを束ねたブロックレベルでサイクリックな配置に並列転送でもって動的に配置しなおす処理過程と、各プロセッサに配置されたブロックについてLU分解を行う処理過程と、LU分解した結果について、ブロックレベルのサイクリックなデータ配置から行列を均等に列ベクトル方向に分割した配置を介して、行列を行ベクトル方向に分割した配置になるように各プロセッサ間でデータを並列に転送し、並列に前進／後進代入の処理を行う処理過程とを有することを特徴とするメモリ分散型並列計算機による連立1次方程式計算処理方法。

【請求項4】 複数のプロセッサを備え、各プロセッサ間でデータ転送を行うことのできるメモリ分散型並列計算機(1)であって、係数行列を各プロセッサに分配して、ブロック化した外積型のLU分解法により連立1次方程式を解くメモリ分散型並列計算機において、LU分解の対象となる行列を、各プロセッサに対して、列ベクトルを束ねたブロックレベルでサイクリックな配置になるように並列転送により配置しなおすデータ再配置処理手段(2)と、LU分解における行列積の計算の際に、行列積の計算対象となるデータを分割して各プロセッサに転送し、分割したデータに対する各プロセッサにおける行列積の計算と、他の分割したデータについての次の行列積の計算に用いる部分の並列転送とを同時に実行する処理を繰り返すことにより全体の計算を行うLU分解処理手段(3)と、LU分解した結果について、ブロックレベルのサイクリックなデータ配置から行列を均等に列ベクトル方向に分割した配置を介して、行列を行ベクトル方向に分割した配置に変え、並列に前進／後進代入の処理を行う前進／後進代入処理手段(4)とを備えたことを特徴とする連立1次方程式を解くメモリ分散型並列計算機。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、複数のプロセッサ間で通信を行って処理を進めるマルチプロセッサシステムにより、高速に連立1次方程式を解くことができるようにしたメモリ分散型並列計算機による連立1次方程式計算処理方法および計算機に関する。

【0002】連立1次方程式を高速に解く技術は、計算機の利用技術として非常に重要である。特に、高並列計算機により効率よく解く場合には、単なる数学的手法にとどまらず並列性を活かして、高並列計算機の特性を最大限に利用することのできる技術が必要になる。

【0003】

【従来の技術】並列処理向きの連立1次方程式を解くアルゴリズムとして、ブロック化した外積型のLU分解法が知られている。図26は、そのブロック化した外積型のLU分解法の概略を説明するための図である。

【0004】外積形式のガウスの消去法をブロック化した方法で、図26に示す配列AをLU分解する。ブロック幅をdとする。この方法では以下のような処理を行う。k番目の処理で、更新部分A^(k)を次の計算で更新する。

【0005】

$$A^{(k)} = A^{(k)} - L_2^{(k)} \cdot U_2^{(k)} \quad \dots\dots(1)$$

k+1番目の処理では、A^(k)をブロック幅dで分解して、dだけ小さいマトリックスを同じ式で更新する。

【0006】L₂^(k)、U₂^(k)は以下の式で求める必要がある。式(1)で更新を行う際に、

$$B^{(k)} = ((L_1^{(k)})^T, (L_2^{(k)})^T)^T U_1^{(k)}$$

と分解し、

$$U2^{(k)} = (L1^{(k)})^{-1}U2^{(k)}$$

と更新する。

【0007】このようなブロック化した外積型のLU分解法をメモリ分散型並列計算機で実行する場合には、各プロセッサの負荷ができるだけ均等になるように、各プロセッサのメモリにデータを効率よく分配し、また各プロセッサ間で処理対象データの交換を効率よく行う必要がある。しかしながら、従来、ユーザインタフェースの簡略化などの面から、ブロック化されたデータを各プロセッサに順番に配置することが考えられているだけであり、必ずしも各プロセッサのLU分解の負荷が均等になってはいなかった。また、プロセッサ間でのデータ通信も並列性が十分でないため、通信コストの増大を招いていた。

【0008】

【発明が解決しようとする課題】大規模な連立1次方程式を解くには、CPUの性能と大規模なメモリシステムが必要である。メモリ分散型のマルチプロセッサで連立1次方程式を高速に解くには、各プロセッサのメモリにデータを効率よく配置することと、効率のよいデータの転送を行う方式を考える必要がある。

【0009】また、問題を解くユーザインタフェース（ホストのアプリケーションインタフェース）を煩雑にせずに実現する必要がある。本発明は上記問題点の解決を図り、性能を引き出すために負荷を分散する最適なデータ配置を動的に行い、データの転送時間が少なくかつ転送を計算と同時にを行う方式を提供することを目的とする。

【0010】

【課題を解決するための手段】図1は本発明の原理説明図である。本発明は、上記の問題を解決するため、連立1次方程式の解法の1つである外積形式のガウスの消去法をブロック化した方法を、以下のように実現する。

【0011】メモリ分散型並列計算機1は、複数のプロセッサを備え、任意の2つのプロセッサが直接通信を行うことができる計算機である。

① メモリ分散型並列計算機1におけるデータ再配置処理手段2は、列ベクトルを束ねたブロックを各プロセッサ（以下、PEという）に分散して配置しなおす。この配置に必要なデータ転送を、並列に行うことにより高速化する。

【0012】② LU分解処理手段3は、ブロックのLU分解を行う途中で行列積の計算を行うとき、データを各PEに転送する。このときデータを分割して転送し、分割したデータに対する計算を各PEで行い、これを繰り返すことにより全体の計算を行う。ここで、最初の転送時間が少なく、かつ以降の転送が計算と同時にできる方法を用い、実際の転送時間が計算時間と重なることにより非常に短くなったように見えるようにする。

【0013】③ 前進／後進代入処理手段4は、LU分解したデータを前進／後進代入を行って並列に効率よく解くために、ブロックレベルのサイクリックなデータ配置から行列を均等に列ベクトル方向に分割した配置を介して、行列を行ベクトル方向に分割した配置に並べ換え、前進／後進代入を実行する。

【0014】

【作用】LU分解を効率よく並列に実行するために、実際の計算を行う前に行列を列ベクトル方向に分割して配置していたものを、ブロックレベルでサイクリックな配置に動的に並列に配置しなおす。

【0015】LU分解を行う上での行列積を効率よく並列に行うために、各プロセッサのベクトル処理と並列実行をバランスさせ、計算を行うのに必要な転送を、見かけ上、1つのプロセッサから1つのプロセッサへの転送に要する時間程度で行なえるようにする。

【0016】ブロックレベルでサイクリックに並べ換えたデータ配置でLU分解を行った後、ブロックレベルのサイクリックなデータ配置から行列を均等に列ベクトル方向に分割した配置を介して、行列を行ベクトル方向に分割した配置に変える。この転送を並列に行う。その結果について、並列に前進／後進代入の処理を行って解く。

【0017】以上のように、データを並列転送で再配置することにより、LU分解の負荷を均等にし、1対nプロセッサ間通信のコストを見かけ上、1対1のプロセッサ間通信のコストに下げ、前進／後進代入処理における方程式を並列に解くことができるようにデータを並列転送で再配置する。

【0018】

【実施例】以下、本発明の実施例を図を用いて説明する。図2は本発明の実施例に係るメモリ分散型並列計算機の例、図3は図2に示すプロセッサ（PE）の構成を示す図である。

【0019】本発明は、例えば図2に示すようなハードウェアを持つメモリ分散型並列計算機によって実現される。各プロセッサ（PE）10はクロスバーネットワーク15に接続され、それぞれスカラ演算を行うスカラユニット11と、ベクトル演算を行うベクトルユニット12と、プログラムの命令列および演算対象データを記憶する主記憶装置13と、任意の他のプロセッサとの間でクロスバーネットワーク15を介して通信を行うPE間通信ユニット14とからなる。

【0020】各プロセッサ10は、例えば図3に示すように構成され、スカラユニット11は、主記憶装置13のデータを一時的に保持するキャッシュメモリ21、演算に用いる汎用レジスタ／浮動小数点レジスタ22、スカラ命令を実行するスカラ演算機23などからなる。主記憶装置13からフェッチした命令がベクトル命令であるときには、ベクトルユニット12が起動される。ベク

トルユニット12は、主記憶装置13からデータをロードするためのロードパイプライン24、主記憶装置13へデータをストアするためのストアパイプライン25、ベクトル演算対象の一連のデータを保持するベクトルレジスタ26、特定の演算対象データをマスクするマスクレジスタ27、演算対象データを指定するマスクパイプライン28、ベクトルデータの乗算を行う乗算パイプライン29、ベクトルデータの加減算または論理演算を実行する加算/論理演算パイプライン30、ベクトルデータの除算を行う除算パイプライン31を備える。

【0021】次に、本発明により連立1次方程式を解く方式について詳細に説明する。

〔1〕動的にデータを並べ換える方法

初めに、動的にデータを並べ換える方法について説明する。

【0022】図2に示すようなメモリ分散型並列計算機において、データは分散されて配置されている。二次元配列の場合、列方向の部分に分割して各プロセッサ（PE）10に割り当てられる。この二次元配列の行列をある幅を持ったブロックを集めたものと考えて、このブロックを並べ換える。

【0023】 $A(n, m) = A(n, 1:d) + A(n, d+1:2d) + A(n, 2d+1:3d) + \dots$
 $= A_1 + A_2 + A_3 + \dots + A_k$

ただし、 $A_j = A(n, (j-1)*d+1:j*d)$
 これを以下のように並べ換える。プロセッサ数を $\#pe$ とする（プロセッサ i （ $i=1, \dots, \#pe$ ））。

【0024】ブロック A_j を $\text{mod}(j-1, \#pe) + 1$ となるプロセッサに割り付ける。 $\text{mod}(a, b)$ は整数 a を整数 b で割ったときの剰余を表す。配列 A と同じ大きさの配列 B を、同じように各プロセッサに分散して割り付ける。図2に示すメモリ分散型並列計算機では、各プロセッサはクロスバーネットワーク15に結合されていて、同時に転送を行うことができる。また、同じプロセッサに対して同時に読み込みと書き込みができる。この機能を使って上記の並べ換えを図4に示すような手順で行う。

【0025】図4は本発明の実施例における並べ換えの処理フローを示す。各プロセッサ（PE）に、ブロック A_j を順番に同じ数だけ並べて A とする。ブロックの総数 j_t はプロセッサ数 $\#pe$ で割り切れるように並べる。1つのプロセッサにあるブロックの数 $\#b$ を、 $\#b = j_t / \#pe$ とする。

【0026】並べ換えの処理では、まず図4に示すステップS1において、 $\text{count} = 0$ とし、 $\text{mod}(n_1 * \#b, \#pe) = 0$ である最小の正の整数 n_1 を探す。次に、ステップS2において、各PEのPE番号を pno （ $pno = 1, \dots, \#pe$ ）としたとき、各PEで $k = 1 + (pno - 1) / n_1$ とする。

【0027】ステップS3において、各PEで次の計算をする。

$p = (pno - 1) * \#b + k$
 $p1 = (p - 1) / \#pe$
 $p2 = \text{mod}(p - 1, \#pe)$
 $q = p2 * \#b + p1 + 1$

ステップS4において、 $B_q = A_p$ の転送を各PEで行う。ここで A_p は各PEにあり、 B_q は各々異なったPEになるため、転送は完全に並列に行うことができる。

【0028】ステップS5において、 $\text{count} = \text{count} + 1$ とする。ステップS6において、 $\text{count} > \#b$ であるかどうかを判定する。 $\text{count} > \#b$ であれば、この処理を終了し、 $\text{count} > \#b$ でなければステップS7の処理を行う。

【0029】ステップS7において、各PEで次の計算をする。

$k = k + 1$
 $k = \text{mod}(k - 1, \#b) + 1$

この後、ステップS3へ戻り、同様に処理を繰り返す。

【0030】図5は実施例におけるブロックの転送例を示す。図5の例では、 $\#pe = 4$ 、 $\#b = 4$ である。

A 、 B における1つの矩形は1ブロックを表し、各ブロック内の数字は説明のためのブロックの番号を表す。図4の処理フローに示すように、 $\text{mod}(\#b, \#pe) = 0$ のため、 $n_1 = 1$ となり、PE1において $k = 1$ 、PE2において $k = 2$ 、PE3において $k = 3$ 、PE4において $k = 4$ となる。したがって、1回目の転送パスでは、PE1の配列 A の1番目のブロック1はPE1の配列 B の1番目に、PE2の配列 A の2番目のブロック6はPE2の配列 B の2番目に、PE3の配列 A の3番目のブロック11はPE3の配列 B の3番目に、PE4の配列 A の4番目のブロック16はPE4の配列 B の4番目に転送される。

【0031】続いて、2回目の転送パスにおいて、 $k = 2$ （PE1）、 $k = 3$ （PE2）、 $k = 4$ （PE3）、 $k = 1$ （PE4）となり、PE1の配列 A の2番目のブロック2はPE2の配列 B の1番目に、PE2の配列 A の3番目のブロック7はPE3の配列 B の2番目に、PE3の配列 A の4番目のブロック12はPE4の配列 B の3番目に、PE4の配列 A の1番目のブロック13はPE1の配列 B の4番目に転送される。

【0032】同様に、3回目の転送パスにおいて、 $k = 3$ （PE1）、 $k = 4$ （PE2）、 $k = 1$ （PE3）、 $k = 2$ （PE4）となり、PE1の配列 A の3番目のブロック3はPE3の配列 B の1番目に、PE2の配列 A の4番目のブロック8はPE4の配列 B の2番目に、PE3の配列 A の1番目のブロック9はPE1の配列 B の3番目に、PE4の配列 A の2番目のブロック14はPE2の配列 B の4番目に転送される。

【0033】同様に、4回目の転送パスにおいて、 $k =$

4 (PE1), $k=1$ (PE2), $k=2$ (PE3), $k=3$ (PE4) となり, PE1の配列Aの4番目のブロック4はPE4の配列Bの1番目に, PE2の配列Aの1番目のブロック5はPE1の配列Bの2番目に, PE3の配列Aの2番目のブロック10はPE2の配列Bの3番目に, PE4の配列Aの3番目のブロック15はPE3の配列Bの4番目に転送される。

【0034】以上のように転送してデータを並べ換えることにより, 1つのPEにおいて, 同時に複数の読み込みまたは同時に複数の書き込みが起きるような衝突がなく, かつ同じPEでは同時に1つの読み込みと1つの書き込みができるので, 転送は完全に並列に行われるとともに, 衝突による待ち合わせが生じることはない。

【0035】〔2〕行列積の効率的な方法

前述のようにして並べ換えたものについて, ブロック化したLU分解を行う方法を以下に説明する。図6は, 実施例におけるLU分解の対象となる行列の例を示す。図26で説明したように外積形式のガウスの消去法をブロック化した方法でLU分解を実行する。そのため, 図6に示す更新部分Uについて, $U = U - C \times R$ の計算を行って, Uを更新する。

【0036】この計算では, 行列Cを各PEに転送する必要がある。考え方として簡単な方法は, 単に行列C全体を各PEに転送する方法である。行列Cの部分を各PEに転送してから行列積の計算を行う場合, 行列C全体を2分木の方法により各PEに2の巾乗のパターンで転送を行う方法が考えられる。すなわち, PE1から残りの $\#pe-1$ 個のPEに転送することを考えた場合, 次のように転送する。

【0037】①PE1からPE2へ行列Cを転送する。②次に, PE1からPE3への転送と, PE2からPE4への転送を同時に行う。③次に, PE1からPE5へ, PE2からPE6へ, PE3からPE7へ, PE4からPE8への転送を同時に行う。このような転送を続けると, 全体の転送コストは $\text{LOG}_2(\#pe)$ のオーダーとなる。

【0038】本実施例では, この全体の転送コストを削減するために, 行列Cを行方向に分割して計算を行う方法を採用する。以下にその方法を説明する。ブロック化したLU分解のk番目のステージで, A_k についてLU分解を行ったあと, 上記のような行列積を行う。このとき, 行列Cをn個に分割する。それを順に $C_1, C_2, C_3, \dots, C_n$ とする。nは, $\#pe/n > 1$ で, $\text{LOG}_2(\#pe/n) < n$ となるように決める。

【0039】図7に示すように, 8個のPE1~PE8で4分割した行列 $C_1 \sim C_4$ について計算を行う場合を例に説明する。PE1がブロックR1のデータを, PE2がブロックR2のデータを, ..., PE8がブロックR8のデータを保持していたとする。各PEの行列 R_i と行列 C_j の積の行列積で部分的に更新を行う。図7に示

すハッチングの部分は, 第1回目の更新を行う部分を表す。

【0040】各PEの行列 R_i と行列積を行う行列 C_j は, PE1からPE8まで順に表すと, 次のとおりである。

1回目の計算: $C_1, C_2, C_3, C_4, C_1, C_2, C_3, C_4$

2回目の計算: $C_4, C_1, C_2, C_3, C_4, C_1, C_2, C_3$

3回目の計算: $C_3, C_4, C_1, C_2, C_3, C_4, C_1, C_2$

4回目の計算: $C_2, C_3, C_4, C_1, C_2, C_3, C_4, C_1$

このような計算を行うためには, C_k の転送を行う必要がある。このため, 次のように転送を行う。

【0041】1回目の転送でブロックkがあるプロセッサは $p = \text{mod}(k-1, \#pe) + 1$ である。このプロセッサpから, 行列Cをn分割した C_i を $\text{mod}(p-2+i, \#pe) + 1$ へ転送する。2回目からは, 1回目の転送で転送されたn個のプロセッサのデータを並列に残りのプロセッサに転送する。順次t回目の転送では $2 \times (t-1) \times n$ (なお, \times は巾乗を表す) のデータを使って並列に転送する。このようにして各プロセッサに C_i を転送する。

【0042】図8は, PE1に行列Cがあった場合の転送例を示す。図8に示すように, 1回目の転送で C_1 はPE1へ, C_2 はPE2へ転送される。2回目の転送では, C_1 はPE1からPE3へ, 同時に C_2 はPE2からPE4へ並列に転送される。次の転送では, C_1 はPE1からPE5へ, PE3からPE7へ転送され, 同時に C_2 はPE2からPE6へ, PE4からPE8へ並列に転送される。

【0043】ここで, 2回目の計算に必要なデータは, 1回目の計算を行っている間に, 別の領域に転送しておくことにより, 転送と計算とを同時に行うことができる。図9および図10は, 行列Cがブロックkつまり $p = \text{mod}(k-1, \#pe) + 1$ にあったときでnが偶数の場合の処理フローを示す。図9および図10に示す処理フローチャートでは, 1回目の計算, つまり奇数回目の計算では第1のワーク領域(W1), 偶数回目の計算では第2のワーク領域(W2)を用いて, 計算を行う。

【0044】まず, ステップS21において, C_i ($i=1, \dots, n$) をプロセッサ $\text{mod}(p+i-2, \#pe) + 1$ のワーク領域W1に転送し, $\$e=0$ とする。ステップS22において, $\$n = N \times 2 \times \e , $\$t = \min(\$n, \#pe - \$n)$ とする。 \min は, 最小値を得る関数である。

【0045】ステップS23において, $\$t$ 個のプロセッサから C_i をワーク領域W1に転送する。また, $s =$

$\text{mod}(p+j-2, \#pe)+1$, $d=\text{mod}(p+t+j-2, \#pe)+1$ とし、プロセッサsからプロセッサdに $j=1, \dots, t$ の t 個を並列に転送する。

【0046】ステップS24において、 $e=e+1$ とする。ステップS25において、 $n>t$ かどうかを判定し、 n が t より大きければステップS22へ戻り、 n が t より小さければステップS26の処理を行う(図10)。

【0047】ステップS26において、 $ct=1$ とする。ステップS27において、 $ct==1$ であるかどうかを判定する。 $ct==1$ であればステップS29へ進み、 $ct==1$ でなければステップS28の処理を行う。

【0048】ステップS28において、後述するステップS33の処理の終了を待って、プロセッサpについてのみ、 C_i ($i=ct$)をデータとしてワーク領域W1へ転送する。

【0049】ステップS29において、プロセッサiからプロセッサ $\text{mod}(I, \#pe)+1$ にデータを転送する(W1からW2への転送)。ステップS30において、開始後、各プロセッサにある C_i のデータで対応する部分の行列の更新を並列に行う(W1を使って計算)。

【0050】ステップS31において、 $ct>1$ であるかどうかを判定する。 $ct>1$ であればステップS33の処理へ進み、 $ct>1$ でなければステップS32の処理を行う。

【0051】ステップS32において、ステップS29の処理(W1からW2への転送)の終了を待つ。ステップS33において、 $ct=ct+1$ とする。プロセッサpについてのみ、 C_i ($i=ct$)をデータとしてワーク領域W2へ転送する。

【0052】ステップS34において、 $ct==n$ であるかどうかを判定する。 $ct==n$ であればステップS36の処理へ進み、 $ct==n$ でなければステップS35の処理を行う。

【0053】ステップS35において、プロセッサiからプロセッサ $\text{mod}(I, \#pe)+1$ にデータを転送する(W2からW1への転送)。ステップS36において、ワーク領域W2のデータを使って対応する行列の更新を各プロセッサで並列に行う。

【0054】ステップS37において、 $ct=ct+1$ とする。ステップS38において、 $ct>n$ であるかどうかを判定する。 $ct>n$ であれば処理を終了し、 $ct>n$ でなければステップS27の処理へ戻る。

【0055】行列Cの分割方法について行方向の分割を説明したが、列方向に分割しても同様に処理を行うことができる。ただし、並列化とベクトル化のバランスを考えると、適当なベクトル長でブロック幅を持たせること

のできる行方向での分割のほうが好ましい。

【0056】この効果は、行列C全体を2分木の方法で、各PEに転送した場合、 $\sim \text{LOG}_2(\#pe)$ のオーダの転送時間がかかるのに対して、 $\sim 1 - (\text{LOG}_2(\#pe/n))/n$ のオーダとなり $\#pe$ 数が大きいときは、非常に高速である。

【0057】

〔3〕前進/後進代入を並列に行う上での方式
LU分解を行った後での前進/後進代入にも高速化のためには並列性が必要である。この並列性を引き出すために、次のように行う。

【0058】第1に、LU分解を行うときに図5に示すようにブロックを各PEに対してサイクリックに割り当てているので、これを元の割り付け方法に戻す。次に、元の行列を列方向に分割していたのを、行方向に分割したデータ配置に変更し、この配置をもとに前進/後進代入を並列に行う。

【0059】すなわち、LU分解を行うときには、図11(A)の行列Aのように、列ベクトルを各PEに分散して配置している。これを前進/後進代入を並列に実行できるように、図11(B)の行列Bのような配置に変更し、行ベクトルを各PEに配置する。これを並列に実行して並べ換える。

【0060】この変換を並列に行うために、行列を各PEに分散配置される境界で図12に示すように分割して、

$$A = (a_{ij}) \quad [i=1, \dots, \#pe, j=1, \dots, \#pe]$$

とする。なお、 $\#pe$ はプロセッサ数であり、図12はプロセッサ数 $\#pe$ が5である場合を示している。

【0061】行ベクトルを各PEに割り付けた行列Aと、同じ大きさの行列を列ベクトルで割り付けた行列Bとの間で、並べ換えのためのデータ転送を、図12に示すハッチング部分のような、対角方向のブロック要素について行う。

【0062】前述のように、図2に示す本実施例のメモリ分散型並列計算機では、各プロセッサに対して同時に1つの読み込みと1つの書き込みが可能である。図13に、図11に示す行列Aから行列Bへの変換の処理フローチャートを示す。

【0063】図13のステップS41において、各プロセッサ(1~ $\#pe$)で、 k =プロセッサ番号、 $j=k$ 、 $\#ct=1$ とする。ステップS42において、並列に各プロセッサで $B_{jk}=A_{jk}$ のデータの配置替えを行う。

【0064】ステップS43において、 $k=\text{mod}(k, \#pe)+1$ 、 $\#ct=\#ct+1$ とする。ステップS44において、 $\#ct>\#pe$ であるかどうかを判定する。 $\#ct>\#pe$ であれば処理を終了し、 $\#ct>\#pe$ でなければステップS42の処理へ戻る。

【0065】行列Bは、図14に示すように、行方向に分割配置されている。ここで、行列BはLU分解できたとする。LU $x=d$ を解くとき、Ly=dを解き、U $x=y$ を順に解く。これを並列に行うために、各PEにd、x、yを重複して持つ。U $x=y$ についても同様に行うことができるので、Ly=dについて説明する。

【0066】まず、PE1で $L_{11} \times y_1 = d_1$ を解く。PE1の y_1 を各プロセッサ上の変数yの y_1 の部分へ2の中乗パターンで転送する。(#pe-1) 個のPEで並列に $d_i = d_i - L_{i1} \times y_1$ を行う ($i=2, \dots, \#pe$)。

【0067】同様に、PE2で $L_{22} \times y_2 = d_2$ を解く。PE2の y_2 を各プロセッサ上の変数yの y_2 の部分へ2の中乗パターンで転送する。(#pe-2) 個のPEで並列に $d_i = d_i - L_{i2} \times y_2$ を行う ($i=3, \dots, \#pe$)。

【0068】同様に、PEkで $L_{kk} \times y_k = d_k$ を解く。PEkの y_k を各プロセッサ上の変数yの y_k の部分へ2の中乗パターンで転送する。(#pe-k) 個のPEで並列に $d_i = d_i - L_{ik} \times y_k$ を行う ($i=k, \dots, \#pe$)。

【0069】最後に、 $L_{55} \times y_5 = d_5$ を解いて、 y_5 を各プロセッサ上の変数yの y_5 の部分へ2の中乗パターンで転送する。結果として、各プロセッサに解yが求まる。

【0070】次に、 200×200 の行列を5プロセッサで解く場合を例にして、本発明の適用例を詳しく説明する。ブロック幅を10と仮定する。すなわち、この例では全部で20ブロック \times 20ブロックの行列となっている。

【0071】この行列を各PEに配置すると、プロセッサ数が5であるので、各PEが担当する部分はそれぞれ20ブロック \div 5の4ブロックの列となる。これらのブロックを、図15(A)に示すように、PE1から順にブロック1、ブロック2、ブロック3、 \dots 、ブロック20とする。

【0072】並列に計算を実行する部分を各PEに均等に割り付けるために、図15(A)に示すデータ配置を、図15(B)に示すように並べ換える。ここでは、各ブロックをブロック・サイクリック (block cyclic) に並べ換えている。この並べ換えでは、例えばブロック2、6、10、14、18の転送を同時に行い、読み込みと書き込みとが各PEでそれぞれ行われるようにして、並列転送を実現する。

【0073】並べ換えの結果、図15(B)に示すように、PE1の行列はブロック1、6、11、16の並びとなり、以下PE2はブロック2、7、12、17、PE3はブロック3、8、13、18、PE4はブロック4、9、14、19、PE5はブロック5、10、15、20となる。

【0074】最初に、ブロック1をLU分解する。ブロック1 (図16のb1) は、PE1だけで計算する。図16のハッチング部分が計算完了となる。図16はブロックb1と各PEの転送先となるワーク領域の関係を示す。各PEは第1および第2のワーク領域を持つ。PE1の第1ワーク領域をW11、第2ワーク領域をW12、PE2の第1ワーク領域をW21、第2ワーク領域をW22、PE3の第1ワーク領域をW31、第2ワーク領域をW32、PE4の第1ワーク領域をW41、第2ワーク領域をW42、PE5の第1ワーク領域をW51、第2ワーク領域をW52とする。

【0075】図16にハッチングで示したブロックb1をC1 \sim C3に3等分した場合を考える。この部分の計算が完了したならば、C1をPE1のW11へ、C2をPE2のW21へ、C3をPE3のW31へそれぞれ転送する。次に、その結果を使って、W11 (C1) のデータをPE4のW41へ、W21 (C2) のデータをPE5のW51へ並列転送する。

【0076】行列積の計算では、初めに各PEの第1のワーク領域 (W11、W21、W31、W41、W51) に格納されたCiを使って計算を行う。図17に示すハッチング部分が最初に計算する部分である。PE1においてC1 \times R1の行列積が、PE2においてC2 \times R2の行列積が、PE3においてC3 \times R3の行列積が、PE4においてC1 \times R4の行列積が、PE5においてC2 \times R5の行列積がそれぞれ計算されることになる。

【0077】これらの計算と同時にオーバーラップしてW11からW22へ、W21からW32へ、W31からW42へ、W41からW52へ、並列にデータ転送を行うとともに、PE1が保持するブロックb1のC3をW12に転送する。

【0078】次に、各PEの第2のワーク領域 (W12、W22、W32、W42、W52) に格納されたCiを使って計算を行う。図18に示すハッチング部分が次に計算する部分である。PE1においてC3 \times R1の行列積が、PE2においてC1 \times R2の行列積が、PE3においてC2 \times R3の行列積が、PE4においてC3 \times R4の行列積が、PE5においてC1 \times R5の行列積がそれぞれ計算されることになる。

【0079】これらの計算と同時にオーバーラップしてW12からW21へ、W22からW31へ、W32からW41へ、W42からW51へ、並列にデータ転送を行うとともに、PE1が保持するブロックb1のC2をW11に転送する。

【0080】3回目の計算では、各PEの第1のワーク領域 (W11、W21、W31、W41、W51) に格納されたCiを使って計算を行う。図19に示すハッチング部分が計算する部分である。PE1においてC2 \times R1の行列積が、PE2においてC3 \times R2の行列積

が、PE3において $C1 \times R3$ の行列積が、PE4において $C2 \times R4$ の行列積が、PE5において $C3 \times R5$ の行列積がそれぞれ計算されることになる。

【0081】図15(B)に示すブロック2(図20のb2)に関するLU分解と対応する行列積の計算は、図20(A)に示すPE2のハッチング部分をLU分解し、計算に必要な部分をブロックb1と同じようにC1~C3に3等分して転送することにより行う。

【0082】図20(B)はブロックb2と各PEの転送先となるワーク領域の関係を示す。C1をPE2のW21へ、C2をPE3のW31へ、C3をPE4のW41へそれぞれ転送する。次にその結果を使って、W21(C1)のデータをPE5のW51へ、W31(C2)のデータをPE1のW11へ並列転送する。

【0083】次に、各PEの第1のワーク領域(W11, W21, W31, W41, W51)に格納されたCiを使って計算を行う。図21に示すハッチング部分が今回計算する部分である。PE1において $C2 \times R1$ の行列積が、PE2において $C1 \times R2$ の行列積が、PE3において $C2 \times R3$ の行列積が、PE4において $C3 \times R4$ の行列積が、PE5において $C1 \times R5$ の行列積がそれぞれ計算されることになる。

【0084】これらの計算と同時にW21からW32へ、W31からW42へ、W41からW52へ、W51からW12へ並列にデータ転送を行うとともに、PE2が保持するブロックb2のC3をW22に転送する。

【0085】以下、図18および図19を用いて説明したブロックb1の場合と同様に計算と転送を行い、ブロックb2についての計算が終了すると、図15(B)に示すブロック3(図22のb3)に関するLU分解と対応する行列積の計算を行う。図22(A)に示すブロックb3をLU分解したあと、図22(B)に示すようにブロックb3をC1~C3に3等分し、各PEのワーク領域への転送を行う。そして、ブロックb3に関しても、図17ないし図19で説明したのと同様に計算と転送を行い、以下、ブロック4, 5, ..., 20まで同様に処理してLU分解を完了する。

【0086】全てのブロックについてLU分解を行った後、最終処理として、サイクリックに並べ換えたブロックを元の並びに戻す。図23は、図15に示すようにブロック・サイクリックに並べ換えた行列を元の並びに戻す例を示している。例えばブロック2, 6, 10, 14, 18に着目すると、図23(A)から(B)への転送例から明らかなように、これらのブロックは同時に転送が可能である。

【0087】次に列方向に分割して配置していたものを行方向に分割する並びに並べ換えて、並列に前進/後退代入を行う。図24は、ブロック化された行列の対角ブロック方向の要素ブロックに着目して、並列転送で並べ換える例を示す。PE1の A_{51} をPE5の1番目の領域

へ、PE2の A_{12} をPE1の2番目の領域へ、……、と、いように並列転送を行って、ブロックを並べ換える。ブロックの並べ換えにより、配置が変わったところで、前進/後退代入を行う。

【0088】図25は、行方向に分割された行列と各PEの関係を示す。解ベクトルを求めるための b_1, b_2, b_3, b_4, b_5 は各PEで重複して保持する。計算が終了した時点で各PEが解ベクトルを持つことになる。計算手順は以下のとおりである。

【0089】(1) $L_{11}x_1 = b_1$ をPE1で解く。

(2) x_1 を b_1 の領域に求めて、各PEへ転送する。
 $b_1 = x_1$

(3) $i > 1$ なるPEiで $b_i = b_i - L_{i1} \times x_1$ を計算する。

【0090】(4) 次にPE2で、 $L_{22}x_2 = b_2$ を解いて x_2 を求める。

(5) x_2 を各PEへ転送する。 $b_2 = x_2$

(6) $i > 2$ なるPEiで $b_i = b_i - L_{i2} \times x_2$ を計算する。

【0091】以下、同様に繰り返して前進代入を終了する。前進代入のあと、後退代入も同様に行う。

【0092】

【発明の効果】以上説明したように、本発明によれば、次のような効果がある。

① データの並べ換えを動的に行うことにより、並列に実行する部分を各プロセッサに均等に割り付けることができるようになり、並列に実行する上での効率が向上する。実際に、ブロック化した外積型のガウスの消去法をもとにしたLU分解の行列積の部分に対しては、行列を列ベクトル方向に均等に分割して計算した場合の実効性能は、ハードウェア性能の6割5分程度である。これは計算過程が進むと行列積で更新する部分が小さくなり、配置されているプロセッサ数が急激に減少するので、並列効率が悪くなるためである。

【0093】これに対して、データをサイクリックに分割すると、つまり列ベクトルを束ねたブロックに番号を振り、その番号をiとしたとき、i番目のブロックが、 $\text{mod}(I-1, \#pe) + 1$ 番目のプロセッサ($\#pe$ はプロセッサ数)に割り付けられるように配置した場合には、実行性能は、ハードウェア性能の9割~9割5分程度が達成される。

【0094】② 行列積の部分の計算方法に関しては、行列積部分の計算に必要なデータを分割して、転送・計算する。このとき、本発明によれば転送の大部分を計算と同時に行うことができ、計算と同時に行うことのできない最初の転送時間だけが見かけ上の転送時間となる。この転送も並列に行う工夫により、例えば2分木転送に比べて、大幅に転送時間を短縮することができる。この結果、単純に行列積で必要なデータを各プロセッサに2分木のパターンで転送する場合が $\text{LOG}2(\#pe)$ に

比例するのに比べ、転送時間は、転送するデータの分割数を $\#div$ とすると、 $1 + (\text{LOG}2(\#pe/\#div) / \#div)$ のオーダーになり、第2項は0.5以下にすることができる。したがって、2台以上あるシステムで特にプロセッサ数が大きくなった場合に非常に効率がよい。

【0095】③ また、最後に前進／後進代入をこのままの配置で解くと、列ベクトルは1つのプロセッサ上にあるため、前進／後進代入部分の並列性が利用できず、並列化できない。このため、サイクリックなデータの配置を、一度、列ベクトル方向に均等分割する配置に戻し、その後、行ベクトル方向に均等分割する配置に変える。このことにより、前進／後進代入部分を並列に実行することができるようになり、処理時間の大幅な短縮が可能になる。

【図面の簡単な説明】

【図1】本発明の原理説明図である。

【図2】本発明の実施例に係るメモリ分散型並列計算機の例を示す図である。

【図3】図2に示すプロセッサの構成を示す図である。

【図4】本発明の実施例における並べ換えの処理フローを示す図である。

【図5】本発明の実施例におけるブロックの転送例を示す図である。

【図6】本発明の実施例におけるLU分解の対象となる行列の例を示す図である。

【図7】本発明の実施例における行列積の計算を説明する図である。

【図8】PE1に行列Cがあった場合の転送例を示す図である。

【図9】データの転送と行列積の計算の処理フローを示す図である。

【図10】データの転送と行列積の計算の処理フローを示す図である。

【図11】前進／後進代入処理時の並べ換えの例を示す図である。

【図12】前進／後進代入処理時の並べ換えを説明する図である。

【図13】前進／後進代入処理時の並べ換えの処理フローを示す図である。

【図14】前進／後進代入処理を説明する図である。

【図15】本発明の適用例の説明図である。

【図16】本発明の適用例の説明図である。

【図17】本発明の適用例の説明図である。

【図18】本発明の適用例の説明図である。

【図19】本発明の適用例の説明図である。

【図20】本発明の適用例の説明図である。

【図21】本発明の適用例の説明図である。

【図22】本発明の適用例の説明図である。

【図23】本発明の適用例の説明図である。

【図24】本発明の適用例の説明図である。

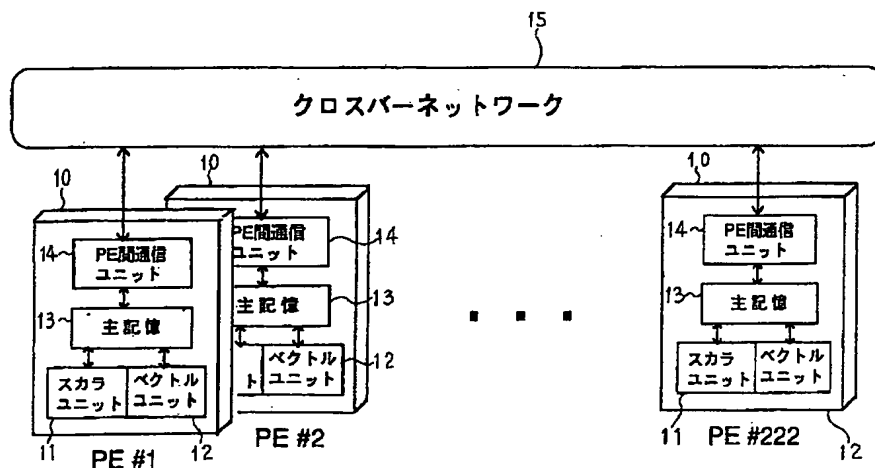
【図25】本発明の適用例の説明図である。

【図26】ブロック化した外積型のLU分解法の説明図である。

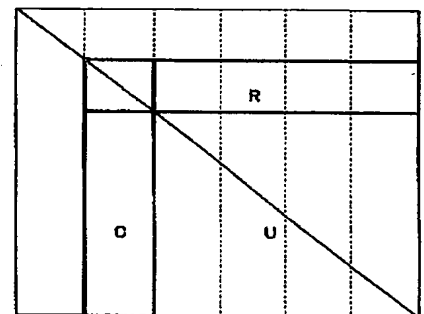
【符号の説明】

- 1 メモリ分散型並列計算機
- 2 データ再配置処理手段
- 3 LU分解処理手段
- 4 前進／後進代入処理手段

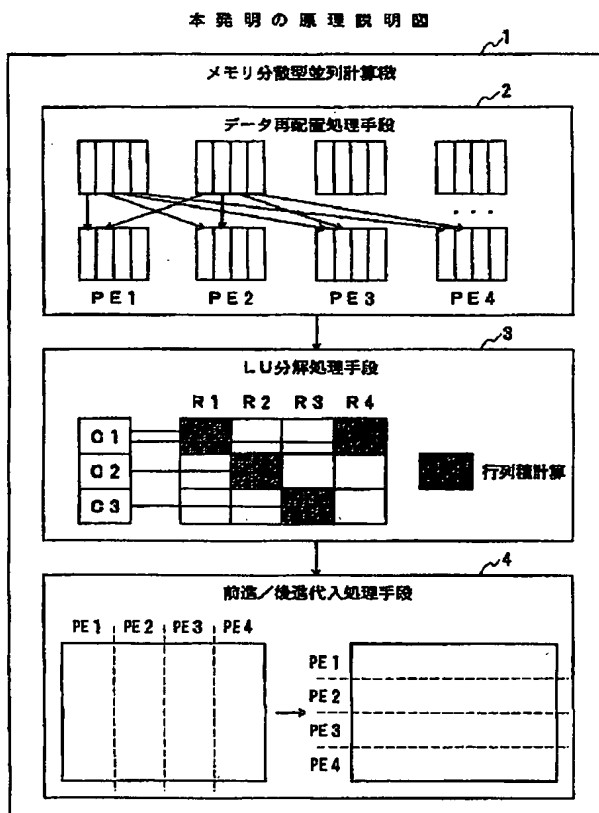
【図2】



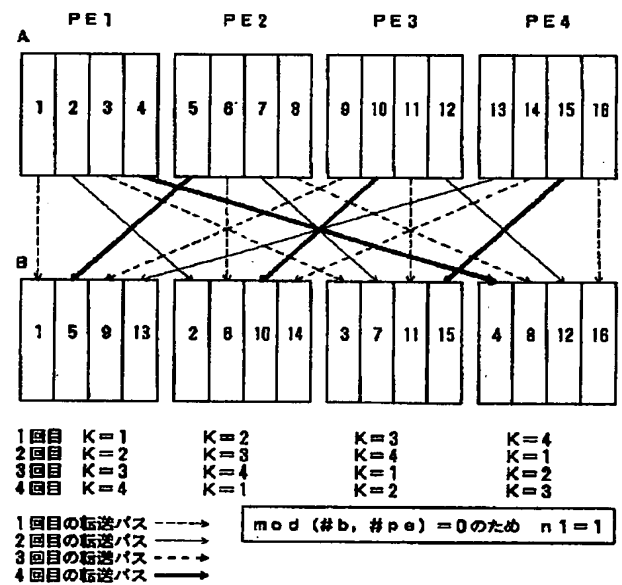
【図6】



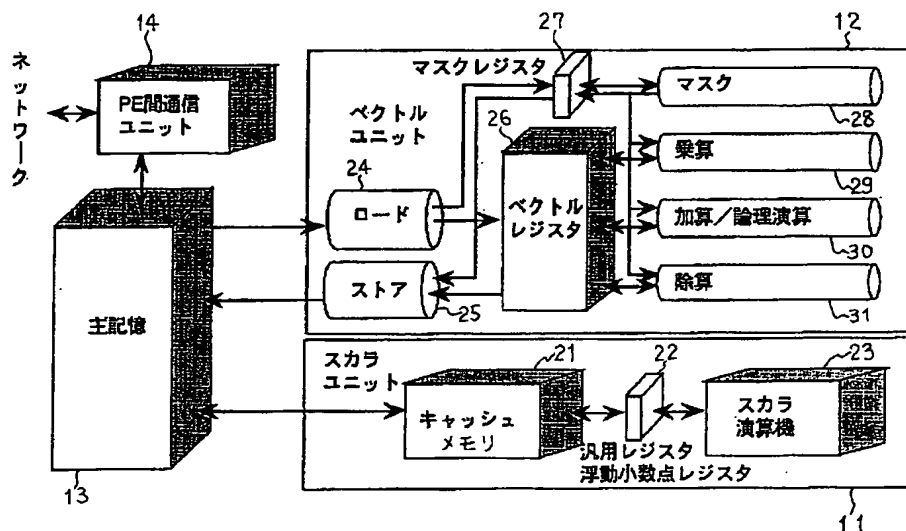
【図1】



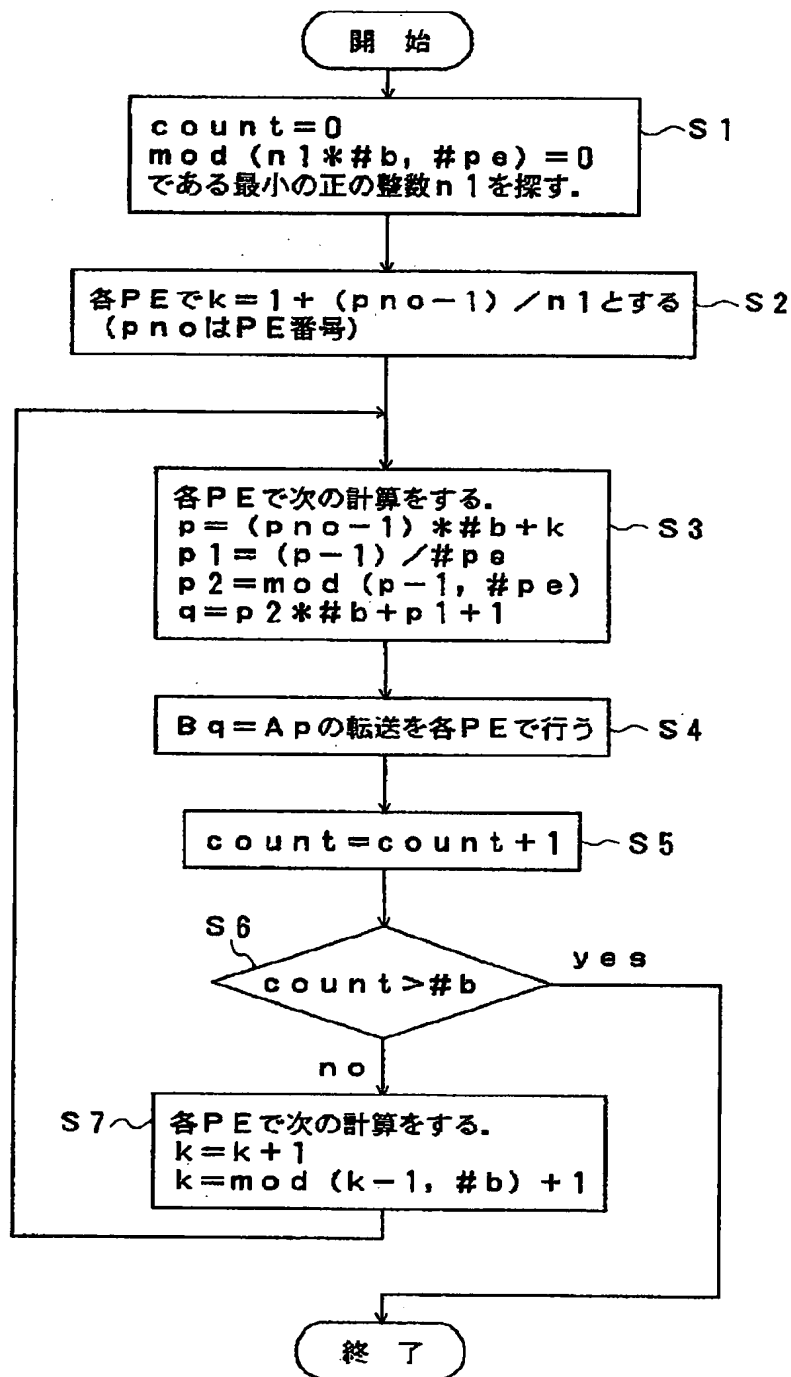
【図5】



【図3】



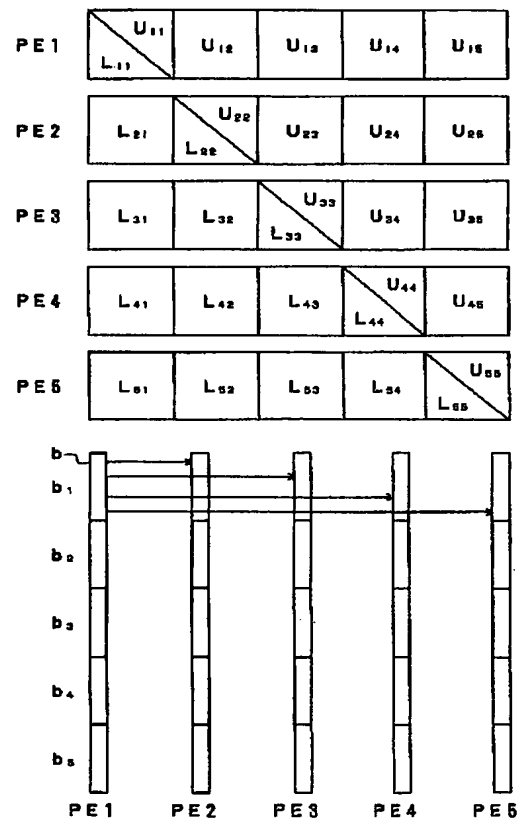
【図4】



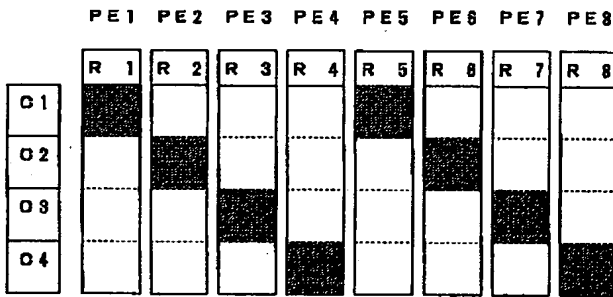
【図12】

A ₁₁	A ₁₂	A ₁₃	A ₁₄	A ₁₅
A ₂₁	A ₂₂	A ₂₃	A ₂₄	A ₂₅
A ₃₁	A ₃₂	A ₃₃	A ₃₄	A ₃₅
A ₄₁	A ₄₂	A ₄₃	A ₄₄	A ₄₅
A ₅₁	A ₅₂	A ₅₃	A ₅₄	A ₅₅

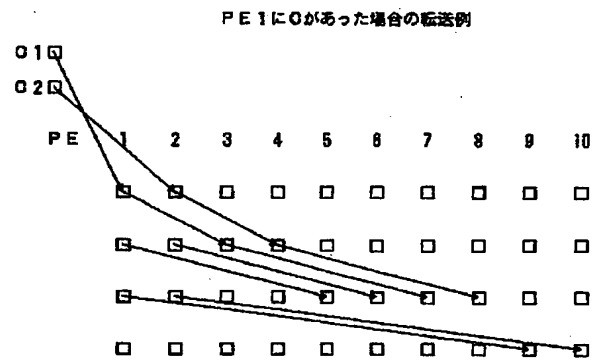
【図25】



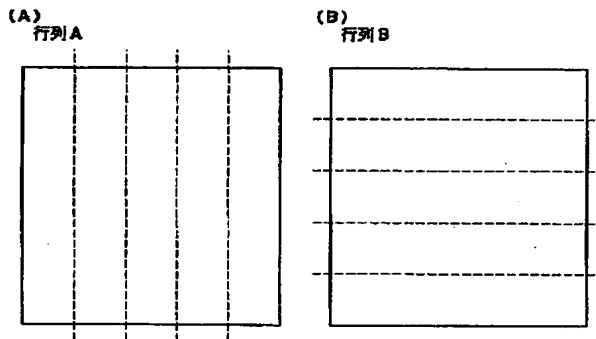
【図7】



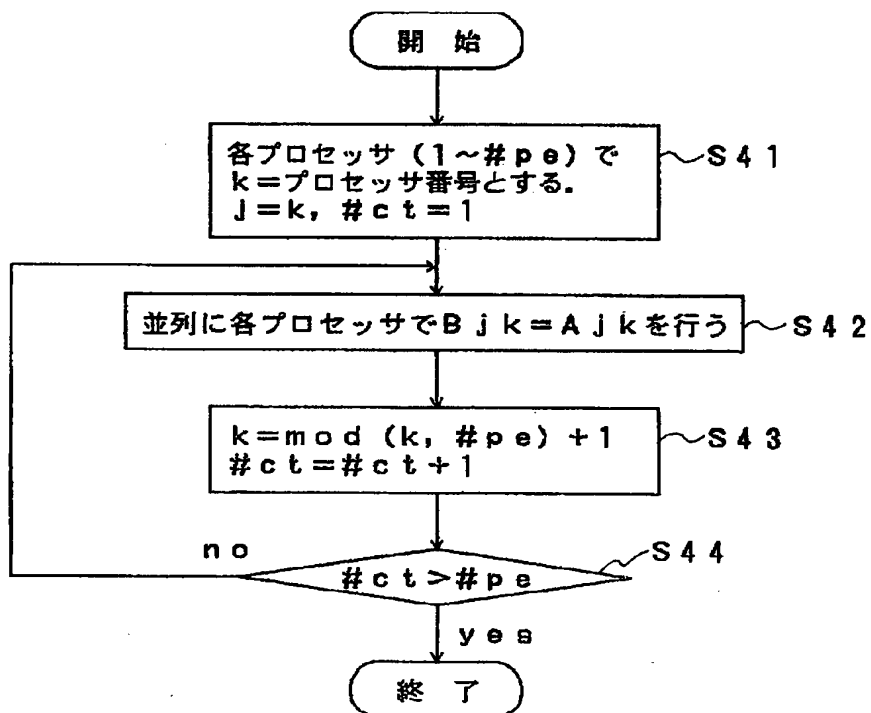
【図8】



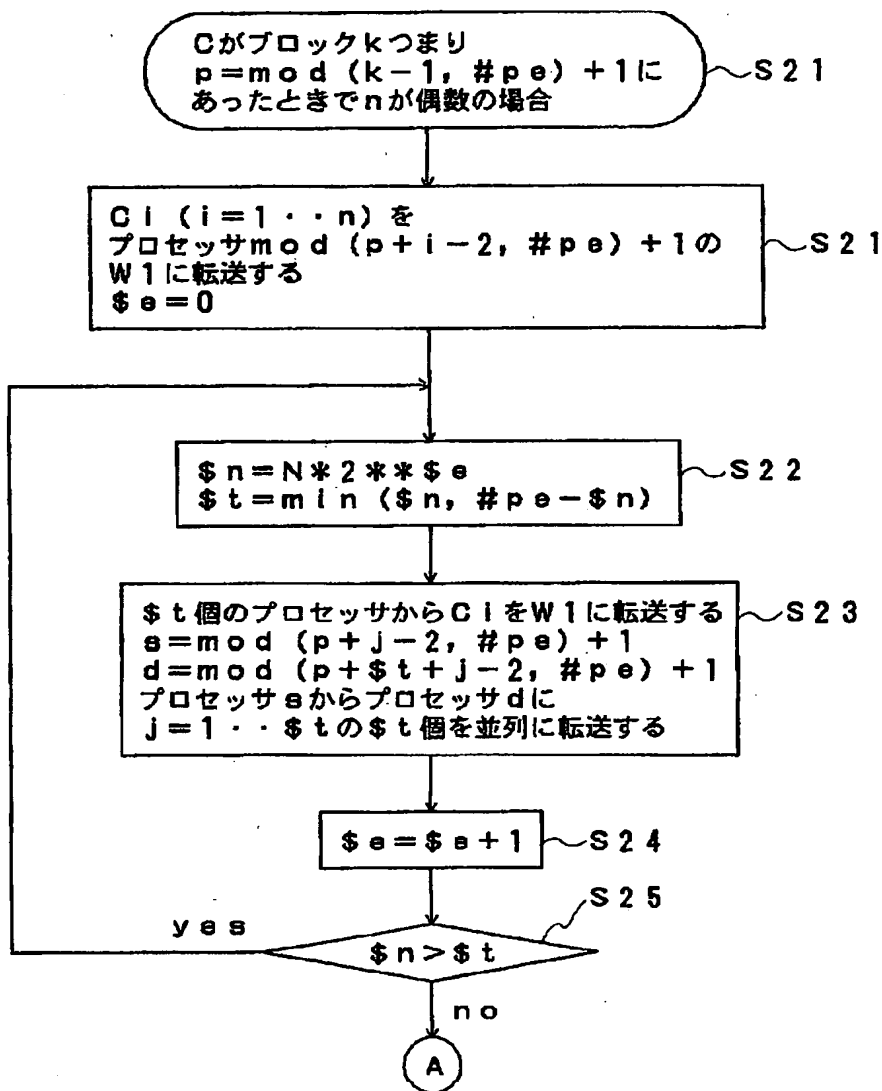
【図11】



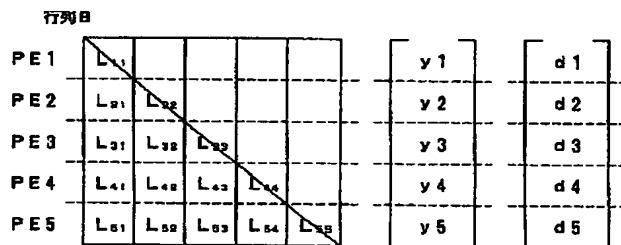
【図13】



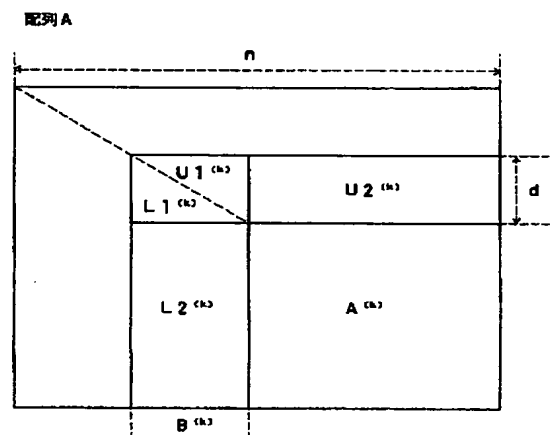
【図9】



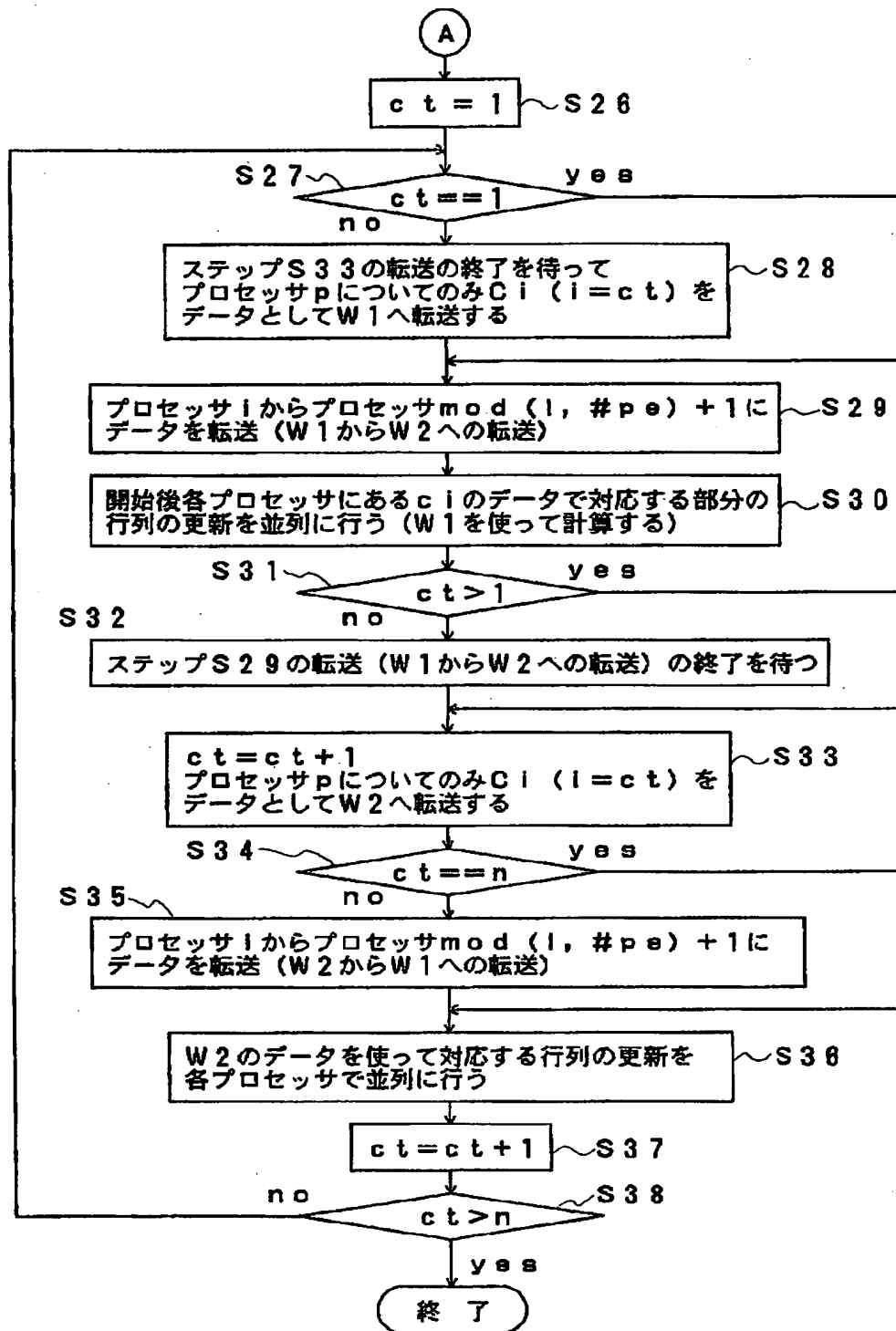
【図14】



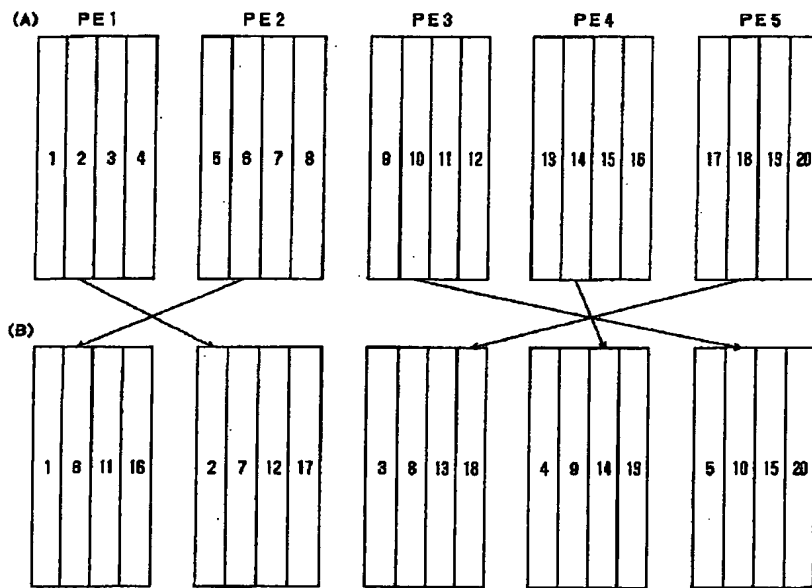
【図26】



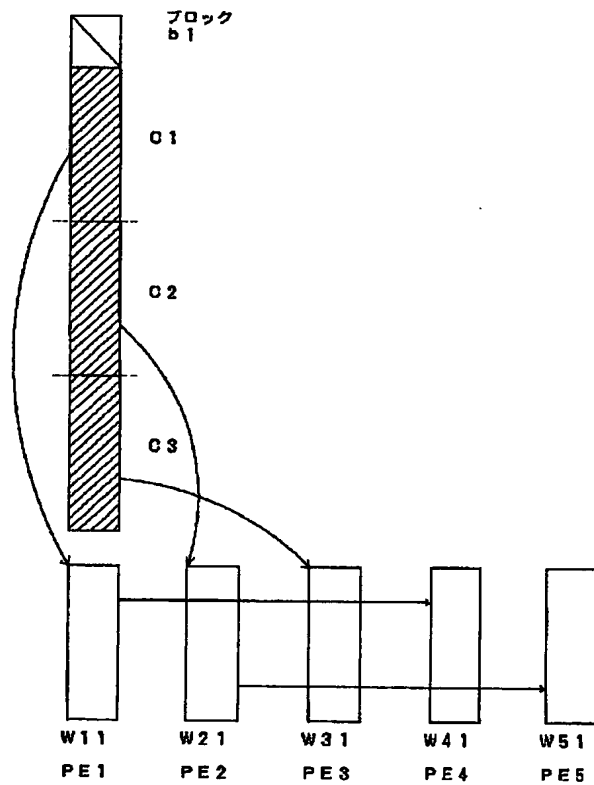
【図10】



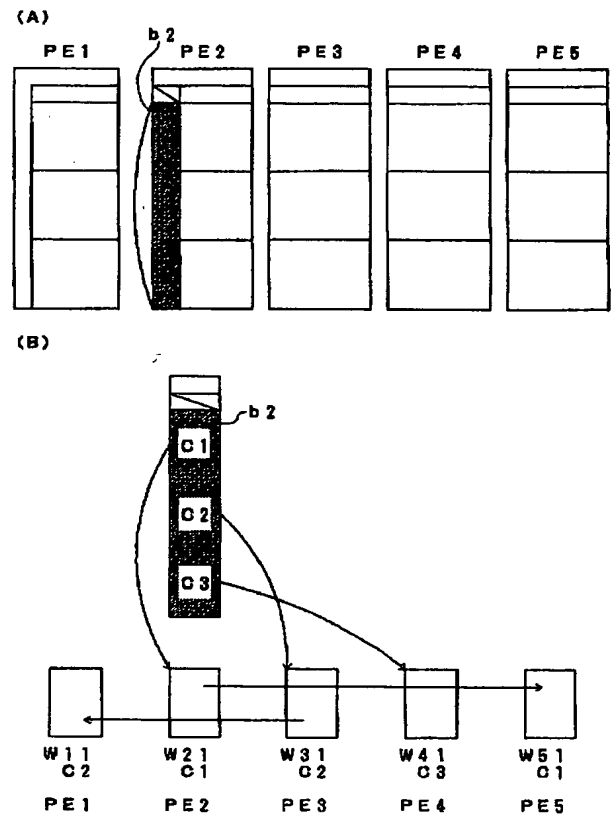
【図15】



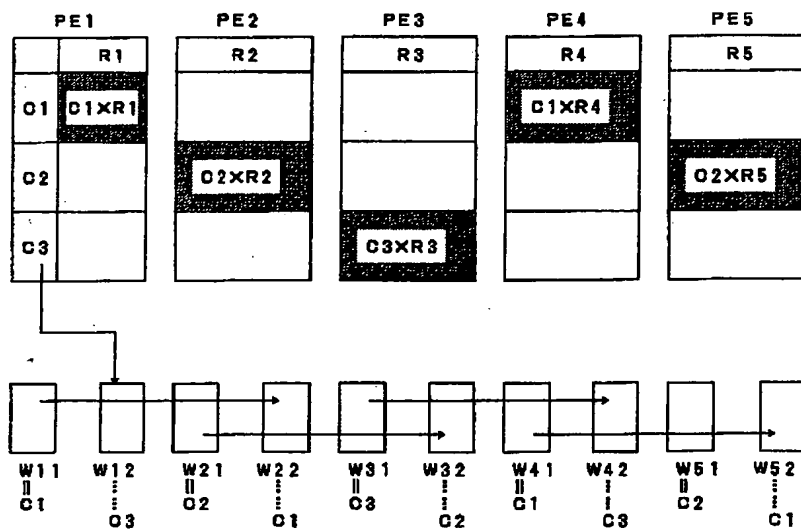
【図16】



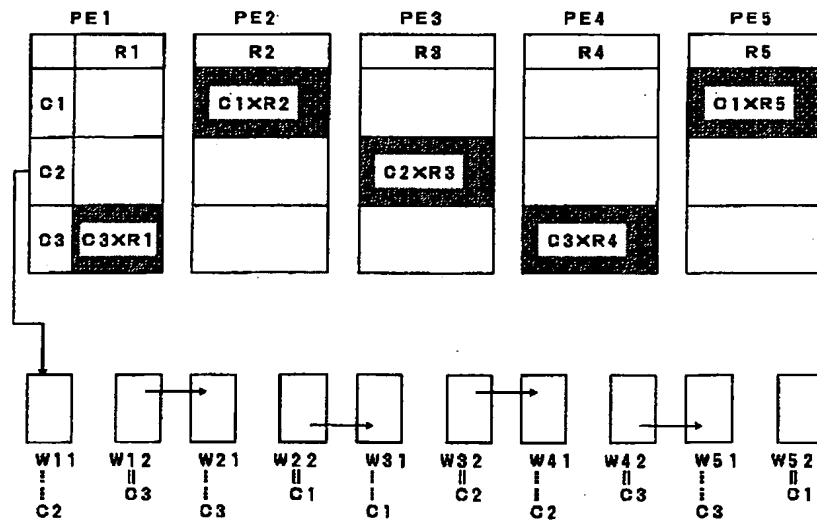
【図20】



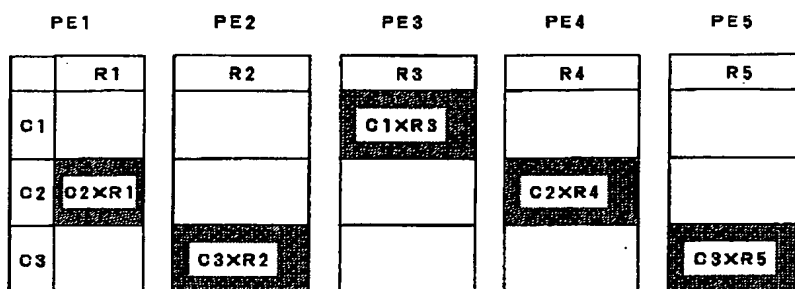
【図17】



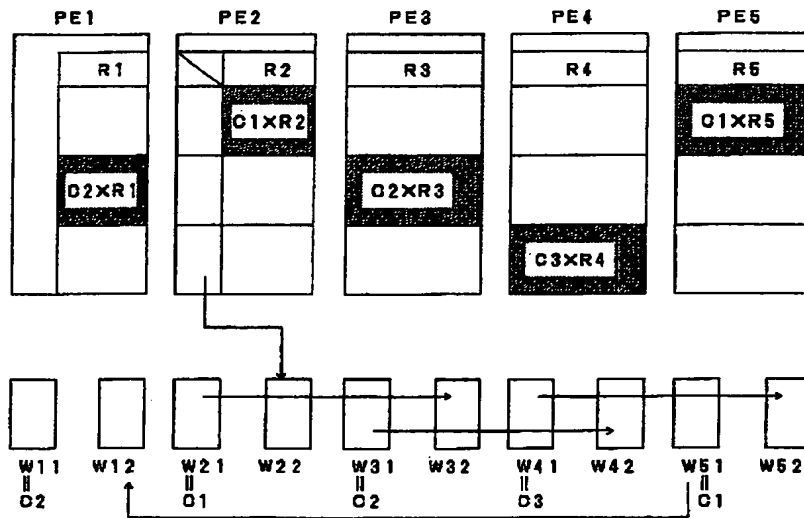
【図18】



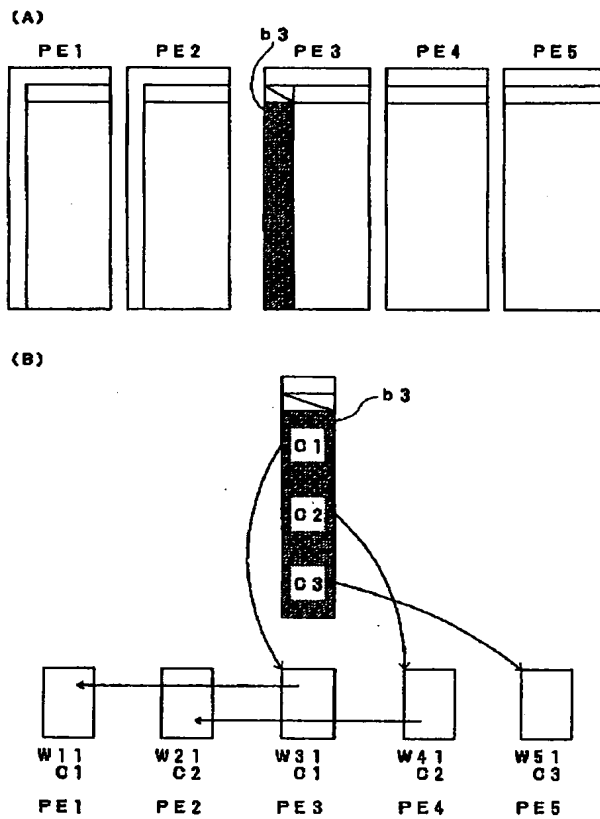
【図19】



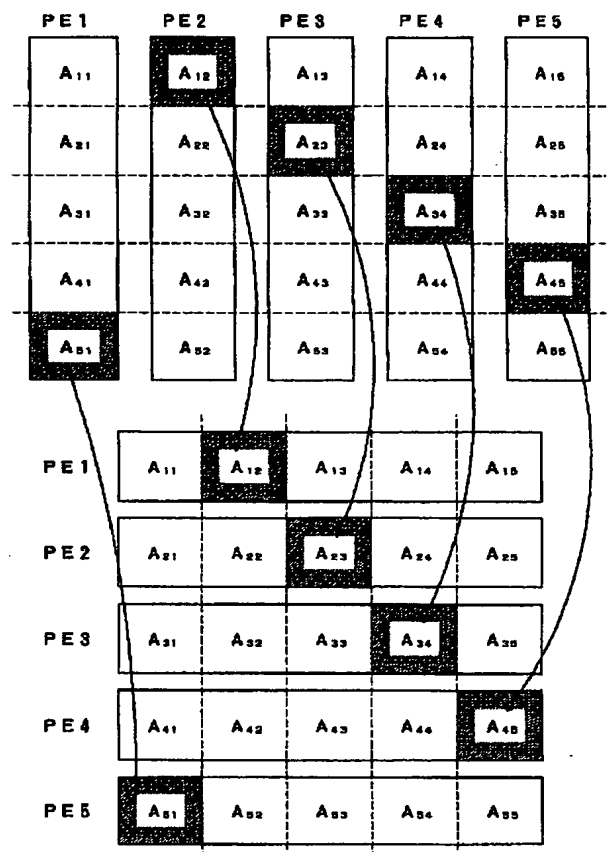
【図21】



【図22】



【図24】



【図23】

